1997

# A Hybrid Immersive / Non-Immersive Virtual Environment Workstation

N96-057
Department of the Navy

Report Number 97247



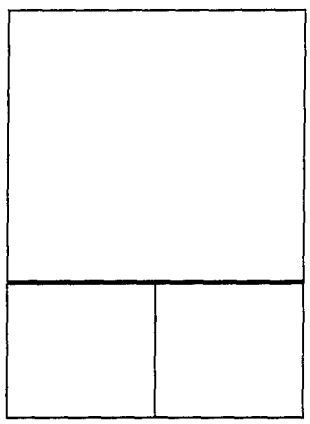
**Submitted by:**
Fakespace, Inc.
241 Polaris Ave.
Mountain View, CA 94043
Phone (415) 688-1940
Fax (415) 688-1949

## Introduction

The implementation of a prototype software application for the Hybrid Immersive / Non-Immersive system has been accomplished. The application, written in C, runs on Silicon Graphics Reality Engines. In order to get the stereo images to both the Immersive Push Viewer and to the large tabletop Non-Immersive display, an option called a Multi-Channel Option (MCO) is used.

## Video Configuration

In order to test the prototype software that has been implemented, two stereo pairs (4 images) need to be generated and displayed. This may be accomplished by using the MCO on the SGI in the 2@640x480_60+1@1280x1024_60 mode which divides up the frame buffer as shown:



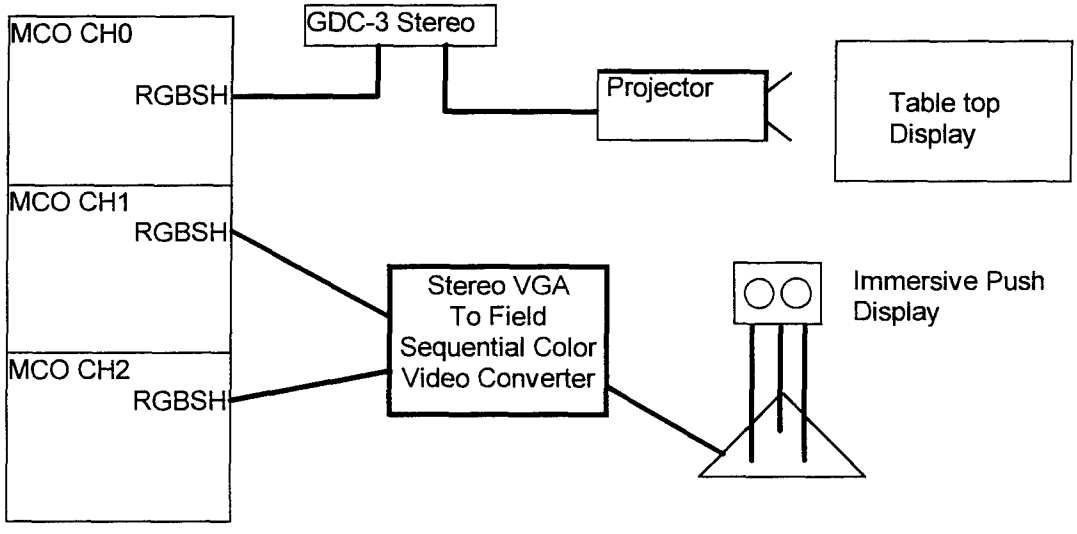
The large upper area is a 1280 by 1024 area of the larger 1280x1504 graphics frame buffer. The lower two areas are 640 by 480 pixels.

The two 640x480_60 images are routed from MCO channels 1 and 2 to the Immersive Push display. This video is not in the correct format and must be converted to the right timing as will be described later.

The 1280x1024_60 image is routed to the projector in the Non-Immersive display. This image is not stereo so it is divided in half. Each half is used for one of the two eye views. This is accomplished by running the video into a set of electronics that adds an additional vertical sync pulse into the middle of the picture. The result is a display with a resolution of 1280x492 at 120 hertz. The user views the display with shutter glasses and each eye sees one of the 1280x492 regions.

The Immersive Push display uses a video format called Field Sequential color. This type of video permits the display to offer high resolution and color images in a small display. For small displays (under 14") the traditional approaches to achieving a color display are limited. Traditionally, color has been achieved by using three colored sub-pixels. On a color LCD or CRT you will generally find sub-pixels of the 3 color primaries red, green, and blue. These sub-pixels are used to create

the color images you see. As you can well imagine, placing a high powered magnifier in front of a display with sub-pixels is going to lead to difficulties because you will see the sub-pixels rather than the overall color image on the display. Field sequential color gets around this problem by displaying the color images over time rather than as colored sub-pixels. The colored fields are presented fast enough that your eye integrates them over time and perceives a full color image. Field sequential video is a relatively new and rather obscure format so it's not generally supported as an output format from computers. The SGI Reality Engine does support field sequential video but not in the combinations we need, so a converter that takes in VGA and produces field sequential color is used.




## Input Devices

The Immersive Push Display uses a serial port to communicate it's position and orientation. The navigation paradigm is implemented on the host from the raw data reported. The navigation paradigm for the Non-Immersive display has not been fully implemented yet. We would like an interface that's consistent with the navigation paradigm used by the Push. In order to deal with this, the first implementation of the prototype software uses the keyboard as the navigational interface for the Non-Immersive user.

## Avatars

The two users can move in the space independently. As a result, each viewer does not know where the other viewer is. This is an issue as they need to know this so that they can discuss particular features of the terrain or situation. An initial solution to this has been to show the location of the Immersive user to the Non-Immersive user by means of an avatar. The Immersive Push Display is represented by a 3-D icon (a model of the Enterprise) in the Non-Immersive user's world. This

helps communicate the location and orientation of the Immersive user to the Non-Immersive user. Interestingly, displaying an avatar for the Non-Immersive user is more difficult to achieve.

The Non-Immersive user does not have a representation that we are all familiar with. Imagine for a moment that you are in a car in unfamiliar terrain and you have someone helping you navigate. The navigator is map reading and has their finger on the map to represent the car's location (actually the *believed* location). What's the navigator's relationship to the car? If you were some how to look out of the window and see a representation of the navigator in the sky, what would be most helpful? This is important because there may be an error in where the navigator *believes* the car is versus where it *actually* is. If there was some way for the driver to verify the location of the navigator with respect to the car, then there would be less chance of the driver getting lost since errors in the current location would be noticed by the driver. Meanwhile, the navigator could be concentrating on route planning as well as where to turn next to achieve an optimal route or get back to an optimal route should local conditions that can't be shown on the map arise, such as a flooded road or other road block.

In this example, we can see how the Immersive (driver) and Non-Immersive (navigator) users have to work together and would ideally have representations for each other in the virtual case.

## Software Structure

The prototype software used as a test bed for exploring the capability of the Immersive / Non-Immersive system is written in C and is based on Performer and VLIB.

# Appendix 1 - The Software Prototype Used

```
/* benchFly.c - interface for joint exploration of a Performer based
 *              space.
 * One user interfaces with a Table top display and another user
 * can explore the same database with a Push or Boom
 *
 * Uses the MCO in 2@640x480_60+1@1280x1024_60 mode
 * Boom or Push will need an FSCONVERT-VGA
 * Interface to Table will need GDC-3 for stereo.
 */

#include <string.h>
#include <stdlib.h>
#include <Performer/pf.h>
#include <Performer/pfutil.h>
#include <Performer/pfdu.h>
#include <Performer/pr.h>
#include <math.h>
#include <GL/gl.h>

#include "pfBench.h"
#include "xboomer.h"

#include "pfSimShare.h"

/**pfBench**/
pfBench *myBench;

/**pfmBoom **/
pfmBoom *myBoom;
pfMatrix lastBoomMat;
pfDCS *enterpriseDCS;
pfDCS *townDCS;

/**pfSimShare **/
pfSimShare *myShare;

/**some nodes**/
pfDCS *scaleDCS, *transDCS;
pfScene *boomScene;
pfScene *benchScene;

typedef struct
{
  float curZoomPos[3];
  float curFactor;
  float startSize;
  float curRotZ;

} zoom_struct;

zoom_struct *zoomVar;

void
UpdateZoom(pfBench *myBench)
{
  pfMatrix mat;
  float size;

  pfBench_GetVirtualLookat( myBench, mat, &size);

  pfMakeRotMat( mat, zoomVar->curRotZ, 0.0f,0.0f,1.0f);

  mat[3][0]=(float)zoomVar->curZoomPos[0];
  mat[3][1]=(float)zoomVar->curZoomPos[1];
  mat[3][2]=(float)zoomVar->curZoomPos[2];

  size=zoomVar->startSize*zoomVar->curFactor;
```

```
  pfBench_SetVirtualLookat( myBench, mat, size);
  /*
  printf("curzoom %f %f %f -%f- %f\n", zoomVar->curZoomPos[0],\
         zoomVar->curZoomPos[1], zoomVar->curZoomPos[2],\
         zoomVar->curFactor, size);
         */

}

static void
Usage (void)
{
    pfNotify(PFNFY_FATAL, PFNFY_USAGE, "Usage: benchFly <model>");
    exit(1);
}

int
ProcessKeybdInput(void)
{
  int  i;
  int  j;
  int  key;
  int  dev;
  int  val;
  int  numDevs;

  pfuEventStream events;

  pfuCollectInput();
  pfuGetEvents(&events);
  numDevs = events.numDevs;

  for (j = 0; j < numDevs; j++)
    {
      dev = events.devQ[j];
      val = events.devVal[j];

      if (events.devCount[dev] > 0)
       {
         switch(dev)
           {
           case PFUDEV_KEYBD:
             for (i = 0; i < events.numKeys; i++)
              {
                key = events.keyQ[i];
                if (events.keyCount[key])
                  {
                    switch(key)
                     {
                     case 27:     /* ESC  exit program */
                       return 0;
                       break;
                       /**Set Zoom factor**/
                     case 49:  /**zoom in**/
                       zoomVar->curFactor *= 0.99;
                       break;
                     case 50: /**zoom out**/
                       zoomVar->curFactor *= 1.01;
                       break;
                     case 51:
                       zoomVar->curZoomPos[1]+= zoomVar->curFactor*\
                         zoomVar->startSize*0.05;
                       break;
                     case 52:
                       zoomVar->curZoomPos[1]-= zoomVar->curFactor*\
                         zoomVar->startSize*0.05;
                       break;
```

```
                    case 53:
                      zoomVar->curZoomPos[0]+= zoomVar->curFactor*\
                        zoomVar->startSize*0.05;
                      break;
                    case 54:
                      zoomVar->curZoomPos[0]-= zoomVar->curFactor*\
                        zoomVar->startSize*0.05;
                      break;
                    case 55:
                      zoomVar->curRotZ+=5.0f;
                      break;

                    default:
                      break;
                    }
                  }
                }
              break;
            default:
              break;
            }
        }
    }
  return 1;
}

#if 0
int
ProcessKeybdInput(void)
{
  int  i;
  int  j;
  int  key;
  int  dev;
  int  val;
  int  numDevs;

  pfuEventStream events;

  pfuCollectInput();
  pfuGetEvents(&events);
  numDevs = events.numDevs;

  for (j = 0; j < numDevs; j++)
    {
      dev = events.devQ[j];
      val = events.devVal[j];

      if (events.devCount[dev] > 0)
       {
         switch(dev)
           {
           case PFUDEV_KEYBD:
             for (i = 0; i < events.numKeys; i++)
              {
                key = events.keyQ[i];
                if (events.keyCount[key])
                  {
                    switch(key)
                     {
                     case 27:      /* ESC  exit program */
                       return 0;
                       break;
                     default:
                       break;
                     }
                  }
```

```
            }
            break;
          default:
            break;
          }
      }
    }
  return 1;
}
#endif

void RUNUPTREE( pfNode *node, pfMatrix *mat)
{
      if ( pfIsOfType( node, pfGetDCSClassType() ) )
      {
            pfMatrix nodeMat;
            pfGetDCSMat( (pfDCS *)node, nodeMat);
            pfPostMultMat( *mat, nodeMat );
            /*          printf("RUNUP !\n");*/
      }
      if ( pfIsOfType( node, pfGetSceneClassType() ) )
            return;

      if ( pfIsOfType( node, pfGetGroupClassType() ) )
      {
            pfGroup *parent;
            parent=pfGetParent(node, 0);
            if (parent!=NULL)
            {
                  RUNUPTREE( (pfNode *)parent, mat);
            }
      }
}

void RUNDOWNTREE( pfNode *node, pfMatrix *mat)
{
      if ( pfIsOfType( node, pfGetDCSClassType() ) )
      {
            pfMatrix nodeMat;
            pfGetDCSMat( (pfDCS *)node, nodeMat);
            pfPreMultMat( *mat, nodeMat );
      }
      if ( pfIsOfType( node, pfGetSceneClassType() ) )
            return;

      {
            pfGroup *parent;
            parent=pfGetParent(node, 0);
            if (parent!=NULL)
            {
                  RUNUPTREE( (pfNode *)parent, mat);
            }
      }
}


int
main (int argc, char *argv[])
{
    pfPipe       *pipe;
    pfPipeWindow    *pw;
    pfNode *model;
    pfLightSource *lt;
    int ExitFlag;

    /************************/
    /* Initialize Performer */
```

```
    pfInitArenas();
    pfInit();
    /*********************************************************/

    zoomVar=pfMalloc( sizeof(zoom_struct), pfGetSharedArena() );


    /*********************************************************/
    /* Get a pfmBoom */
    myBoom=(pfmBoom *)pfmNewBoom(pfGetSharedArena(),"./setup");
    /*********************************************************/

    {

    myBoom->coord.xyz[0]= 2565.5f;
    myBoom->coord.xyz[0]= 2389.2f;
    myBoom->coord.xyz[0]= 5.0f;

    myBoom->lastCoord.xyz[0]= 2565.5f;
    myBoom->lastCoord.xyz[0]= 2389.2f;
    myBoom->lastCoord.xyz[0]= 5.0f;
    }

    /*********************************************************/
    /* Get a pfBench */
    if ( (myBench=pfBench_Create("./bench.config"))==NULL)
      {
       printf("Could not open a pfBench.\n");
       exit(-1);
      }
    /*********************************************************/


    /*********************************************************/
    /* Get a pfSimShare */
    if ( (myShare=pfSimShare_Open( (void *)myBoom, (void *)myBench, ""))==NULL)
      {
       printf("Could not open a pfSimShare.\n");
       exit(-1);
      }
    /*********************************************************/


    /*********************************************************/
    /* Use default multiprocessing mode based on number of
     * processors.
     */
    pfMultiprocess( PFMP_APPCULLDRAW );
    /*********************************************************/

    /*********************************************************/
    /* initiate multi-processing mode set in pfMultiprocess call
     * FORKs for Performer processes,  CULL and DRAW, etc. happen here.
     */
    pfConfig();
    pfuInitUtil();
    /*********************************************************/

    /*********************************************************/
    /* Append to Performer search path, PFPATH, files in
     *     /usr/share/Performer/data */

      pfFilePath(".:/usr/share/Performer/data:../geometry:./geometry:../../geome
try");

    /*********************************************************/
```

```
    /*****************************************************/
    /* Create and configure a window*/
    pipe = pfGetPipe(0);
    pw = pfNewPWin(pipe);
    pfPWinType(pw, PFPWIN_TYPE_X);
    pfPWinName(pw, "Fakespace WorkBench");
    pfPWinMode(pw, PFWIN_NOBORDER, 1);

    /*pfBench_ will configure the size and
      type based on config file settings**/
    pfSimShare_ConfigureWindow(myShare, pw);

    pfOpenPWin(pw);
    /*************************************/

    /*************************************/
    /** Initialize event model for keyboard input**/
    pfuInitInput(pw, PFUINPUT_X);

    /*************************************/
    /** Set some global graphics state**/
    pfOverride(PFSTATE_CULLFACE, 0);
    pfCullFace(PFCF_OFF);
    pfEnable(PFEN_HIGHLIGHTING);
    pfEnable(PFEN_LIGHTING);

    /*************************************/
    /**pfSimShare_ConfigureChannels_ will create the necessary channels**/

    pfSimShare_ConfigureChannels( myShare, pipe );
    pfBench_InitSceneGraph( (pfBench *)myShare->benchObject );
    benchScene= (pfScene *)pfBench_GetNode( (pfBench *)myShare->benchObject,
PFBENCH_SCENE);
    boomScene = pfNewScene();

    /*    scene=pfNewScene();*/
    /**Perform some additional channel configuration**/
    {
      pfChannel *chans[4]={NULL,NULL,NULL,NULL};
      int i;
      pfEarthSky *es;

      pfSimShare_GetChannel( myShare, &chans[0], BOOM_LEFT_EYE_CHAN);
      pfSimShare_GetChannel( myShare, &chans[1], BOOM_RIGHT_EYE_CHAN);
      pfSimShare_GetChannel( myShare, &chans[2], BENCH_LEFT_EYE_CHAN);
      pfSimShare_GetChannel( myShare, &chans[3], BENCH_RIGHT_EYE_CHAN);

      for (i=0;i<4;i++)
       {
         if (chans[i]!=NULL)
           {
             es = pfNewESky();
             pfESkyMode(es, PFES_BUFFER_CLEAR, PFES_FAST);
             pfESkyFog(es, PFES_GENERAL, NULL);
             pfESkyColor(es, PFES_CLEAR, 0.6,0.6,0.6,1.0);
             pfChanESky(chans[i],es);
             pfChanLODAttr(chans[i], PFLOD_SCALE, 0.0);
             pfChanTravMode(chans[i], PFTRAV_CULL,PFCULL_VIEW);

           }
       }
    }
    /*************************************/

    /*************************************/
```

```
    /**Create a scene**/


    scaleDCS=pfNewDCS();
    transDCS=pfNewDCS();
    townDCS=pfNewDCS();

    /**And then there was light**/
    lt=pfNewLSource();
    pfLSourcePos(lt, -10.0f, -100.0f, 100.0f, 0.0f);
    pfLSourceOn(lt);

    pfAddChild( benchScene, lt);
    pfAddChild( boomScene, lt);

    /**Get a model; add it to the virtual space**/
    if ((model = pfdLoadFile(argv[1])) == NULL)
      {
       printf("Could not load geometry.\n");
       pfExit();
       exit(-1);
      }


    pfAddChild( (pfGroup *)pfBench_GetNode( (pfBench *)myShare-
>benchObject,PFBENCH_VIRTUAL), \
                                        townDCS);
    pfAddChild(boomScene, townDCS);

    pfAddChild(townDCS,model);
    {
      pfSphere sph;
      pfMatrix move;
      float mat[4][4],size;

      pfGetNodeBSphere( model, &sph );
      pfMakeIdentMat(move);

      pfMakeTransMat( move, sph.center[0],sph.center[1],0);
      pfBench_SetVirtualLookat( myBench, move, sph.radius*2);

      pfBench_GetVirtualLookat( myBench, mat, &size);
      zoomVar->curFactor=1.0f;
      zoomVar->startSize=size;
      zoomVar->curRotZ=0.0f;
      zoomVar->curZoomPos[0]=sph.center[0];
      zoomVar->curZoomPos[1]=sph.center[1];
      zoomVar->curZoomPos[2]=0.0f;
    }


#if 0
    /**Now setup the scene so that, initially our
      view is positioned and scaled above the model so that
      the model fits within the display surface**/
    {
      pfSphere sph;
      pfGetNodeBSphere( model, &sph );
      printf("center %f %f %f - scale %f\n",\
            -sph.center[0],-sph.center[1],-sph.center[2], sph.radius);


      pfDCSTrans( transDCS, -sph.center[0],-sph.center[1],-sph.center[2]+1.0);
      pfDCSScale( scaleDCS, 40.0/sph.radius );
```

```
    }
#endif


    /*****************************************/
    /**enterprise avatar*/
    {
      pfNode *eModel;
      pfDCS *eSDCS, *eTDCS;

      eSDCS=pfNewDCS();
      eTDCS=pfNewDCS();
      enterpriseDCS=pfNewDCS();
      if ((eModel = pfdLoadFile("enterprise.flt")) == NULL)
       {
         printf("Could not load geometry.\n");
         pfExit();
         exit(-1);
       }
      pfAddChild(townDCS, enterpriseDCS);
      pfAddChild(enterpriseDCS,eSDCS );
      pfAddChild(eSDCS, eTDCS );
      pfAddChild(eTDCS,eModel );

      {
       pfSphere sph;
       pfGetNodeBSphere( eModel, &sph );
       printf("enterprise center %f %f %f - scale %f\n",\
              -sph.center[0],-sph.center[1],-sph.center[2], sph.radius);


       pfDCSTrans( eTDCS, -sph.center[0],-sph.center[1],-sph.center[2]);
       pfDCSScale( eSDCS, 10.0/sph.radius );

      }



    }

    /*****************************************/
    /**miscellaneous*/
    pfPhase(PFPHASE_LOCK);
    pfFrameRate(20.0f);

    /**optimize the database and scene**/
    /*
    pfdMakeSharedScene( benchScene );
    pfdMakeSharedScene( boomScene );
    pfFlatten( scene, 0 );
    */
    /*****************************************/

    /*****************************************/
    /**set table angle**/
    /*    pfBench_SetTableAngle( myBench, 0.0f );*/
    /*****************************************/

    /** Main Loop **/
    ExitFlag=0;
    while (!ExitFlag)
    {
      /**Act on keyboard events**/
      if (ProcessKeybdInput()<1)
       ExitFlag=1;

      /* Go to sleep until next frame time. */
```

```
        pfSync();
        /*************************************/
        /* Do Latency Critical Stuff */
        /* Track the view first*/


        pfmGetBoomMat( ((pfmBoom *)myShare->boomObject), lastBoomMat );
        pfDCSMat( enterpriseDCS, lastBoomMat );
        pfSimShare_UpdateViews(myShare, lastBoomMat, boomScene, benchScene);

        UpdateZoom( (pfBench *)myShare->benchObject );

        /*********************************/

        /* Initate cull/draw for this frame. */
        pfFrame();
        /*************************************/

    }

    /* Terminate parallel processes and exit. */
    pfExit();
    /*************************************/
}

/* This Performer based software is based on software originally
 * developed by SGI.  The required license follows.
 */

/*
 * Copyright 1995, Silicon Graphics, Inc.
 * ALL RIGHTS RESERVED
 *
 * UNPUBLISHED -- Rights reserved under the copyright laws of the United
 * States.   Use of a copyright notice is precautionary only and does not
 * imply publication or disclosure.
 *
 * U.S. GOVERNMENT RESTRICTED RIGHTS LEGEND:
 * Use, duplication or disclosure by the Government is subject to restrictions
 * as set forth in FAR 52.227.19(c)(2) or subparagraph (c)(1)(ii) of the Rights
 * in Technical Data and Computer Software clause at DFARS 252.227-7013 and/or
 * in similar or successor clauses in the FAR, or the DOD or NASA FAR
 * Supplement.  Contractor/manufacturer is Silicon Graphics, Inc.,
 * 2011 N. Shoreline Blvd. Mountain View, CA 94039-7311.
 *
 * Permission to use, copy, modify, distribute, and sell this software
 * and its documentation for any purpose is hereby granted without
 * fee, provided that (i) the above copyright notices and this
 * permission notice appear in all copies of the software and related
 * documentation, and (ii) the name of Silicon Graphics may not be
 * used in any advertising or publicity relating to the software
 * without the specific, prior written permission of Silicon Graphics.
 *
 * THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
 * EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
 * WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.
 *
 * IN NO EVENT SHALL SILICON GRAPHICS BE LIABLE FOR ANY SPECIAL,
 * INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY
 * DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
 * WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY
 * THEORY OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE
 * OR PERFORMANCE OF THIS SOFTWARE.
 *
 */
```